Task3

常见的特征工程包括：

1. 异常处理：
    - 通过箱线图（或 3-Sigma）分析删除异常值；
    - BOX-COX 转换（处理有偏分布）；
    - 长尾截断；
2. 特征归一化/标准化：
    - 标准化（转换为标准正态分布）；
    - 归一化（抓换到 [0,1] 区间）；
    - 针对幂律分布，可以采用公式：$\log(1+x1+\text{median})$
3. 数据分桶：
    - 等频分桶；
    - 等距分桶；
    - Best-KS 分桶（类似利用基尼指数进行二分类）；
    - 卡方分桶；
4. 缺失值处理：
    - 不处理（针对类似 XGBoost 等树模型）；
    - 删除（缺失数据太多）；
    - 插值补全，包括均值/中位数/众数/建模预测/多重插补/压缩感知补全/矩阵补全等；
    - 分箱，缺失值一个箱；
5. 特征构造：
    - 构造统计量特征，报告计数、求和、比例、标准差等；
    - 时间特征，包括相对时间和绝对时间，节假日，双休日等；

- 地理信息，包括分箱，分布编码等方法；
- 非线性变换，包括 log/ 平方/ 根号等；
- 特征组合，特征交叉；
- 仁者见仁，智者见智。

6. 特征筛选
   - 过滤式（filter）：先对数据进行特征选择，然后在训练学习器，常见的方法有 Relief/方差选择发/相关系数法/卡方检验法/互信息法；
   - 包裹式（wrapper）：直接把最终将要使用的学习器的性能作为特征子集的评价准则，常见方法有 LVM（Las Vegas Wrapper）；
   - 嵌入式（embedding）：结合过滤式和包裹式，学习器训练过程中自动进行了特征选择，常见的有 lasso 回归；
7. 降维
   - PCA/ LDA/ ICA；
   - 特征选择也是一种降维

平时也会进行一些特征工程操作，但是这次总结的很全面

```
    iqr = box_scale * (data_ser.quantile(0.75) - data_ser.quantile(0.25))
    val_low = data_ser.quantile(0.25) - iqr
    val_up = data_ser.quantile(0.75) + iqr
    rule_low = (data_ser < val_low)
    rule_up = (data_ser > val_up)
    return (rule_low, rule_up), (val_low, val_up)
```

这部分是计算分位数，第一次听说分位数，解释如下

定义

分位数指的就是连续分布函数中的一个点，这个点对应概率p。若概率0<p<1，随机变量X或它的概率分布的分位数Za，是指满足条件p(X≤Za)=α的实数 [1] 。

常见分类

1.二分位数

对于有限的数集，可以通过把所有观察值高低排序后找出正中间的一个作为中位数。如果观察值有偶数个，则中位数不唯一，通常取最中间的两个数值的平均数作为中位数，即二分位数。
一个数集中最多有一半的数值小于中位数，也最多有一半的数值大于中位数。如果大于和小于中位数的数值个数均少于一半，那么数集中必有若干值等同于中位数。
计算有限个数的数据的二分位数的方法是：把所有的同类数据按照大小的顺序排列。如果数据的个数是奇数，则中间那个数据就是这群数据的中位数；如果数据的个数是偶数，则中间那2个数据的算术平均值就是这群数据的中位数。
2.四分位数
四分位数（Quartile）是统计学中**分位数**的一种，即把所有数值由小到大排列并分成四等份，处于三个分割点位置的数值就是四分位数。
1）**第一四分位数**(Q1)，又称“较小四分位数”，等于该样本中所有数值由小到大排列后第25%的数字；
2）**第二四分位数**(Q2)，又称“中位数”，等于该样本中所有数值由小到大排列后第50%的数字；
3）**第三四分位数**(Q3)，又称“较大四分位数”，等于该样本中所有数值由小到大排列后第75%的数字。
第三四分位数与第一四分位数的差距又称四分位距。
3.百分位数
百分位数，统计学术语，如果将一组数据从小到大排序，并计算相应的累计百分位，则某一百分位所对应数据的值就称为这一百分位的百分位数。运用在教育统计学中，例如表现测验成绩时，称PR值。

异常值处理

```
# 这里我包装了一个异常值处理的代码，可以随便调用。
def outliers_proc(data, col_name, scale=3):
    """
    用于清洗异常值，默认用 box_plot（scale=3）进行清洗
    :param data: 接收 pandas 数据格式
    :param col_name: pandas 列名
    :param scale: 尺度
    :return:
    """

    def box_plot_outliers(data_ser, box_scale):
```

```
    """
    利用箱线图去除异常值
    :param data_ser: 接收 pandas.Series 数据格式
    :param box_scale: 箱线图尺度,
    :return:
    """
    iqr = box_scale * (data_ser.quantile(0.75) - data_ser.quantile(0.25))
    val_low = data_ser.quantile(0.25) - iqr
    val_up = data_ser.quantile(0.75) + iqr
    rule_low = (data_ser < val_low)
    rule_up = (data_ser > val_up)
    return (rule_low, rule_up), (val_low, val_up)

data_n = data.copy()
data_series = data_n[col_name]
rule, value = box_plot_outliers(data_series, box_scale=scale)
index = np.arange(data_series.shape[0])[rule[0] | rule[1]]
print("Delete number is: {}".format(len(index)))
data_n = data_n.drop(index)
data_n.reset_index(drop=True, inplace=True)
print("Now column number is: {}".format(data_n.shape[0]))
index_low = np.arange(data_series.shape[0])[rule[0]]
outliers = data_series.iloc[index_low]
print("Description of data less than the lower bound is:")
print(pd.Series(outliers).describe())
index_up = np.arange(data_series.shape[0])[rule[1]]
outliers = data_series.iloc[index_up]
print("Description of data larger than the upper bound is:")
print(pd.Series(outliers).describe())

fig, ax = plt.subplots(1, 2, figsize=(10, 7))
sns.boxplot(y=data[col_name], data=data, palette="Set1", ax=ax[0])
sns.boxplot(y=data_n[col_name], data=data_n, palette="Set1", ax=ax[1])
return data_n
```

十分有用，留着用做参考

```
# 使用时间：data['creatDate'] - data['regDate']，反应汽车使用时间，一般来说价格与使用时间成反比
# 不过要注意，数据里有时间出错的格式，所以我们需要 errors='coerce'
data['used_time'] = (pd.to_datetime(data['creatDate'], format='%Y%m%d', errors='coerce') -
                     pd.to_datetime(data['regDate'], format='%Y%m%d', errors='coerce')).dt.days
```

不太理解errors="coerce",查了一下

参数：

（1）arg：int，float，str，datetime，list，tuple，1-d数组，Series，DataFrame / dict-like，要转换为日期时间的对象

（2）errors：{'ignore'，'raise'，'coerce'}，默认为'raise'

如果为“ raise”，则无效的解析将引发异常

如果为“coerce”，则将无效解析设置为NaT

如果为“ ignore”，则无效的解析将返回输入

（3）dayfirst：bool，默认为False，

如果arg是str或类似列表，则指定日期解析顺序。

如果为True，则首先解析日期，例如12/10/11解析为2011-10-12。

警告：dayfirst = True并不严格，但更喜欢使用day first进行解析（这是一个已知的错误，基于dateutil的行为）

（4）yearfirst：布尔值，默认为False，

如果arg是str或类似列表，则指定日期解析顺序。

如果True解析日期以年份为第一，则将10/11/12解析为2010-11-12。

如果dayfirst和yearfirst均为True，则在yearfirst之后（与dateutil相同）。

警告：yearfirst = True并不严格，但更喜欢使用year first进行解析（这是一个已知的错误，基于dateutil的行为）。

```python
# 计算某品牌的销售统计量，同学们还可以计算其他特征的统计量
# 这里要以 train 的数据计算统计量
train_gb = train.groupby("brand")
all_info = {}
for kind, kind_data in train_gb:
    info = {}
    kind_data = kind_data[kind_data['price'] > 0]
    info['brand_amount'] = len(kind_data)
    info['brand_price_max'] = kind_data.price.max()
    info['brand_price_median'] = kind_data.price.median()
    info['brand_price_min'] = kind_data.price.min()
    info['brand_price_sum'] = kind_data.price.sum()
    info['brand_price_std'] = kind_data.price.std()
    info['brand_price_average'] = round(kind_data.price.sum() / (len(kind_data) + 1), 2)
    all_info[kind] = info
brand_fe = pd.DataFrame(all_info).T.reset_index().rename(columns={"index": "brand"})
data = data.merge(brand_fe, how='left', on='brand')
```

很有用的处理方法

数据分桶

```python
# 数据分桶 以 power 为例
# 这时候我们的缺失值也进桶了，
# 为什么要做数据分桶呢，原因有很多，= =
# 1. 离散后稀疏向量内积乘法运算速度更快，计算结果也方便存储，容易扩展；
# 2. 离散后的特征对异常值更具鲁棒性，如 age>30 为 1 否则为 0，对于年龄为 200 的也不会对模型造成很大的干扰；
# 3. LR 属于广义线性模型，表达能力有限，经过离散化后，每个变量有单独的权重，这相当于引入了非线性，能够提升模型的表达能力，加大拟合；
# 4. 离散后特征可以进行特征交叉，提升表达能力，由 M+N 个变量编程 M*N 个变量，进一步引入非线形，提升了表达能力；
# 5. 特征离散后模型更稳定，如用户年龄区间，不会因为用户年龄长了一岁就变化

# 当然还有很多原因，LightGBM 在改进 XGBoost 时就增加了数据分桶，增强了模型的泛化性

bin = [i*10 for i in range(31)]
data['power_bin'] = pd.cut(data['power'], bin, labels=False)
data[['power_bin', 'power']].head()
```

引用.cut用法

数据分箱（Data binning，也称为离散组合或数据分桶）是一种数据预处理技术，将原始数据分成小区间，即一个 bin（小箱子），它是一种量子化的形式。

Pandas 实现连续数据的离散化处理主要基于两个函数：

- `pandas.cut` 根据指定分界点对连续数据进行分箱处理
- `pandas.qcut` 根据指定箱子的数量对连续数据进行等宽分箱处理

注：所谓等宽指的是每个箱子中的数据量是相同的。

pd.cut

`pd.cut` 可以指定区间将数字进行划分，以下三个值将数据划分成两个区间（及格或者不及格）：

```
pd.cut(df.Q1, bins=[0, 60, 100])
'''
0     (60, 100]
1       (0, 60]
        ...
98      (0, 60]
99      (0, 60]
Name: Q1, Length: 100, dtype: category
Categories (2, interval[int64]): [(0, 60] < (60, 100]]
'''
```

应用到分组中：

```
df.Q1.groupby(pd.cut(df.Q1, bins=[0, 60, 100])).count()
'''
Q1
(0, 60]      57
(60, 100]    43
Name: Q1, dtype: int64
'''
df.groupby(pd.cut(df.Q1, bins=[0, 60, 100])).count()
```

其他参数：

```
# 不用区间，使用数字做为标签（0，1，2，n）
pd.cut(df.Q1, bins=[0, 60, 100],labels=False)
# 指定标签名
pd.cut(df.Q1, bins=[0, 60, 100],labels=['不及格','及格',])
# 包含最低部分
pd.cut(df.Q1, bins=[0, 60, 100], include_lowest=True)
# 是否包含右边，闭区间，下例 [89, 100)
pd.cut(df.Q1, bins=[0, 89, 100], right=False)
```

pd.qcut

pd.qcut 指定所分箱子的数量，pandas 会自动进行分箱：

```
pd.qcut(df.Q1,q=2)
'''
0      (51.5, 98.0]
1     (0.999, 51.5]
          ...
98    (0.999, 51.5]
99    (0.999, 51.5]
Name: Q1, Length: 100, dtype: category
Categories (2, interval[float64]): [(0.999, 51.5] < (51.5, 98.0]]
'''
```

应用到分组中：

```
df.Q1.groupby(pd.qcut(df.Q1,q=2)).count()
'''
Q1
(0.999, 51.5]    50
(51.5, 98.0]     50
Name: Q1, dtype: int64
'''
df.groupby(pd.qcut(df.Q1,q=2)).max()
```

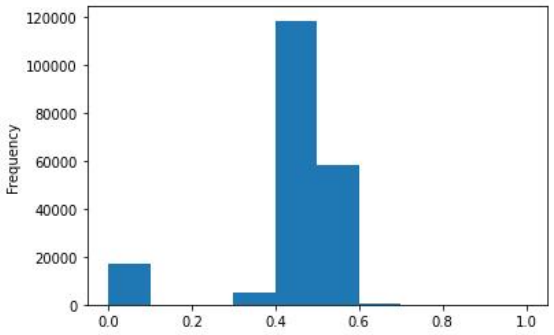
其他参数：

```
pd.qcut(range(5), 4)
pd.qcut(range(5), 4, labels=False)
# 指定标签名
pd.qcut(range(5), 3, labels=["good", "medium", "bad"])
```

```
0    50    100    150    200    250    300    350
```

```
# 我们对其取 log，在做归一化
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
data['power'] = np.log(data['power'] + 1)
data['power'] = ((data['power'] - np.min(data['power'])) / (np.max(data['power']) - np.min(data['power'])))
data['power'].plot.hist()
```

[19]: <AxesSubplot:ylabel='Frequency'>



很标准的归一化

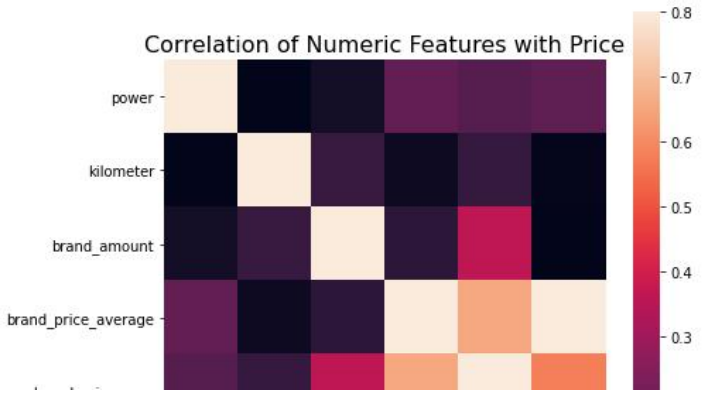
1) 过滤式 ¶

```
# 相关性分析
print(data['power'].corr(data['price'], method='spearman'))
print(data['kilometer'].corr(data['price'], method='spearman'))
print(data['brand_amount'].corr(data['price'], method='spearman'))
print(data['brand_price_average'].corr(data['price'], method='spearman'))
print(data['brand_price_max'].corr(data['price'], method='spearman'))
print(data['brand_price_median'].corr(data['price'], method='spearman'))
```

```
0.5728285196051496

-0.4082569701616764

0.058156610025581514

0.3834909576057687

0.259066833880992

0.38691042393409447
```

```
# 当然也可以直接看图
data_numeric = data[['power', 'kilometer', 'brand_amount', 'brand_price_average',
                     'brand_price_max', 'brand_price_median']]
correlation = data_numeric.corr()

f , ax = plt.subplots(figsize = (7, 7))
plt.title('Correlation of Numeric Features with Price',y=1,size=16)
sns.heatmap(correlation,square = True,  vmax=0.8)
```

```
[27]: <AxesSubplot:title={'center':'Correlation of Numeric Features with Price'}>
```



熟悉的相关性分析，9417中用到了

包裹式

这个感觉文章中写的不够清楚，这里是资料补充

包裹式选择（wrapper）

它与过滤式不同，它会考虑后续的学习器。它会把学习器的性能作为评价准则。

1.LVW

第一步：随机选出特征集；

第二步：计算相关误差，如果误差比原本的小，或者误差相当但是特征数比之前的少，则把子集留下。

2.RFE

第一步：对初始特征进行训练，得到权重

第二步：提出权重最小的特征，构成新的集合

第三步：不断重复，知道满足停止条件为止。

Class sklearn.feature_selection.RFE(estimator,n_features_to_select=None,step=1,estimator_params=None,verbose=0)

参数:

estimator:一个学习器（通常使用SVM和广义线性模型作为estimator）

n_features_to_select:指定要选出几个特征

step:指定每次迭代要剔除权重最小的几个特征

大于等于1：指定每次迭代要剔除权重最小的特征的数量

在0.0~1.0：指定每次迭代要剔除权重最小的特征的比例

estimator_params:一个字典，用于设定estimator的参数

sklearn还提供了RFECV类，它是RFE的一个变体，它执行一个交叉验证来寻找最优的剩余特征数量，因此不需要指定保留多少个特征。原型为

class sklearn.feature_selection.RFECV(estimator,step=1,cv=None,scoring=None,estimator_params=None,verbose=0)

包裹式
主要思想是反复的构建模型（如SVM或者回归模型）然后选择最好的（或者最差的）特征（可以根据系数来选）
把选出来的特征选出来，然后在剩余的特征上重复这个过程，直到所有的特征都遍历过
这个过程特征被消除的次序就是特征的排序，因此这是一种寻找最优特征子集的贪心算法

```
from sklearn.svm import  LinearSVC
from sklearn.datasets import  load_iris
from sklearn.feature_selection import RFE

iris=load_iris()
x=iris.data
y=iris.target
estimator=LinearSVC()
selector=RFE(estimator=estimator,n_features_to_select=2)
selector.fit(x,y)
print('特征数：',selector.n_features_)
print('哪些特征被挑选出来了：',selector.support_)
print('特征评分：',selector.ranking_)#第二，第四个为1，所以被选出来
#特征选择对预测性能的提升没有联系
```

```
特征数： 2
哪些特征被挑选出来了： [False  True False  True]
特征评分： [3 1 2 1]
```

3.4 经验总结¶

特征工程是比赛中最至关重要的的一块，特别的传统的比赛，大家的模型可能都差不多，调参带来的效果增幅是非常有限的，但特征工程的好坏往往会决定了最终的排名和成绩。

特征工程的主要目的还是在于将数据转换为能更好地表示潜在问题的特征，从而提高机器学习的性能。比如，异常值处理是为了去除噪声，填补缺失值可以加入先验知识等。

特征构造也属于特征工程的一部分，其目的是为了增强数据的表达。

有些比赛的特征是匿名特征，这导致我们并不清楚特征相互直接的关联性，这时我们就只有单纯基于特征进行处理，比如装箱，groupby，agg 等这样一些操作进行一些特征统计，此外还可以对特征进行进一步的 log，exp 等变换，或者对多个特征进行四则运算（如上面我们算出的使用时长），多项式组合等然后进行筛选。由于特性的匿名性其实限制了很多对于特征的处理，当然有些时候用 NN 去提取一些特征也会达到意想不到的良好效果。

对于知道特征含义（非匿名）的特征工程，特别是在工业类型比赛中，会基于信号处理，频域提取，丰度，偏度等构建更为有实际意义的特征，这就是结合背景的特征构建，在推荐系统中也是这样的，各种类型点击率统计，各时段统计，加用户属性的统计等等，这样一种特征构建往往要深入分析背后的业务逻辑或者说物理原理，从而才能更好的找到 magic。